Frontend exam – Summary

Some React app learnings

7.11.2023

React

- **React**: a JavaScript SPA UI building library
- renders the output page (HTML+CSS+JS) DOM based on
  - your page template code (public>index.html etc.)
  - Your React components (possibly inside other React components)

    AND
  - the data provided by AJAX (with or without Redux)

JSX

- **JSX**: Basics
  - XML-**like** language mixing JS and React component markup,
  - **not** XML as it's not following XML rules,
  - nor HTML structure rules either
- One syntax example: What is happening in the following?

```
var a =123;
<Xyz abc={{a}}  />
```

  - first go to JS mode with { },
  - then create an object {a}, (thus same as ={{a:a}}
- React components with **C**apitalFirstLetter, HTML elements/attributes with small letter: …
- **className(s)** attribute or so (in React, react-rendered) vs. **class** attribute (HTML, already ready html.
  - Thus, being able to mix JSX and ready HTML, though only okay if unavoidable
- if you **return** JSX, wrap it inside **( )** (((Or make sure to start JSX from same line as the **return** keyword)))

React Hooks: Basics

- Recap also the ES object destructor assignment if needed.
- https://youtu.be/mxK8b99iJTg?t=150 from 02:30 to 19:50, 18 mins
  - (Skip pre-release react installation! no need for: "npm i ...“)
  - (Here also anything with class/constructor you can just follow/skip, only fully understand the hooks version of the same.)
  - Simple output and input UI with react hooks. (No AJAX/persistence here) Pay attention to:
    1. only arrow functions used!
    2. function definitions vs. func calls! (function definitions / ready function objects passed, not function calls)
    3. set state function is created automatically for you
    4. React dev tools used to study component state
    5. Above also Mother passing to Children: data and/or event-handler function objects, in Childrens' props.

  (Of course we would **not** write Todo, TodoForm and App in the same file. This is a basic demo. Also ‘value’/’setValue’ could be called e.g. ‘task’, ‘setTask’. As you can have multiple states in a component, ‘value’ is too ambiguous name)
- https://reactjs.org/docs/hooks-intro.html Tutorial text and examples of the same basic hooks knowledge

Code analysis task (related to previous)

- *What 4-5 things will happen with this?*

```
const [age, setAge] = useState(7);
```

- ***Spoiler alert**: Answer on next slide*

Code analysis task (Answer)

- *What 4-5 things will happen with this:* **const [age, setAge] = useState(7);** *?*
  - *const 'variable' age created to this scope = to this React component*
  - *const 'setAge' created and assigned a setter function for setting new values to age, automagically assigned, no need for our code*
    - *thus we are kind of defining a function called setAge, but we receive the function object from React object created by the useState() call.*
  - *the initial value of age will be 7 (initial value only used when component created)*
  - *useState will return that value 7 and the setAge function as an array*
  - *we use the destructuring assignment to pick that value and assign it to 'age', and that function to 'setAge'*

useState, useEffect, useContext – typical hooks

- https://youtu.be/dpw9EHDh2bM?t=1061 (useState, useEffect, **useContext** introduced)
  - from 17:45 until 54:13 = <36 mins
  - NOTICE, No need to watch the unrelated LATTER PART OF THE SHOW!
- **useState** => used to just setup some fraction of the component's React **state** and its **update function**
- state/props created or changed, render happens => **useEffect( )** fires! Thus useful for attaching any functionality defined inside the useEffect:s
- **useContext** => used to share data / data structures between unrelated React components

useState, useEffect, useContext – typical hooks 2

- https://www.youtube.com/watch?v=TNhaISOUy6Q
- from 0:00 **until 6:58** our 3 Hooks (useState, useEffect, useContext) shown in just 7 mins
- Video progresses fast. Pause and read code!
- The ‘post cleanup’ happens with the function that your useEffect **optionally returns** after it's done with what it wants to do!
- note: **side effects** = things happening outside of the React state/prop change ⇔ re-render -cycle. So e.g. saving something, even crucial to our business process, to backend via AJAX is a "side effect".
- useEffect is an inner function of component function.
  - useEffect has replaced the old React component life-cycle event handlers (Where you could 'attach' your event-handler functions). inner function = function defined inside the component function. Then the visibility of the function is inside the outer function, inside that component only.

useState, useEffect, useContext – typical hooks 3

- https://reactjs.org/docs/hooks-overview.html
- https://reactjs.org/docs/hooks-state.html
- https://reactjs.org/docs/hooks-effect.html Effect Hooks
- https://reactjs.org/docs/context.html#when-to-use-context Context Hooks

General rules of Hooks

- https://reactjs.org/docs/hooks-rules.html
  - Name must start with lower letter text “use”
  - Should only be placed/called in the main level of the react component function
  - Call hooks only from your react functions (e.g. from your other hooks),
    - Do not call them **from** your regular JavaScript functions

Mostly of course the React environment itself calls the hooks, and not us. But if we call, then from our React functions.

How to read this hook code?

- const some = useCustomX(() => { doSomething(kakku); **return ()=> {doSomeAfterStuff();}** } ,**[kukka]**);
  - *Custom hooks too start with lower letter “use”*
  - *Hook only fired **if kukka changes**! Not if any other prop or state changes, like e.g. useEffect does by default.*
  - *After doSomething is done, the second function, **’cleanup function’**, will be executed, the one that calls doSomeAfterStuff*

Context Hook

- Advanced topic, useful still in our case
- https://reactjs.org/docs/hooks-reference.html#usecontext
- https://reactjs.org/docs/context.html#when-to-use-context

- https://www.youtube.com/watch?v=lhMKvyLRWo0

BTW. What happens here? From video above

```
string jsonText = JSON.stringify(myJSObject, null, 2);
```

- Explanation found if you look at the documentation, here with my longer parameter names though:

JSON.stringify(object_to_be_stringified_as_JSON,
null_as_no_replacer_function_needed_here_but_still_needs_something_as_second_param,
white_space_has_to_be_the_*third*_parameter)

Custom Hooks (a bit Advanced topic)

- *https://reactjs.org/docs/hooks-custom.html*
- *custom hooks, like the ones in the second video:*
- *useWindowWidth*
- *useDocumentTitle*

create-react-app – “CRA” OR vite

- **The tool for downloading and setting up the React dev project environment**
- Basic understanding needed
- *What kind of project created?* (npm start => dev time environment with Node server, webpackage/nodemon (and e.g. React Dev tools) starts)
- *What's the relationship with e.g. the /public/index.html and the React app? How the React app starts and builds up the page? index.js | index.html + App.js | and so on.*
- *How is the dev environment related to the build version = when published and put to 'production environment’?*
  - npm build => /build folder with only few mashed up .html and .js (and needed .css plus other static files),
  - no Node anymore, no ES5,6,7,8, no node_modules, just browser runnable ‘DOM API’ JS + html + css + …,
  - served to the client's web browser by even a static web server, from www ports like 80/443 or so

SPA = Single-Page Application

- Only one web page downloaded from the Web Server, but then with JS & AJAX that single page's DOM updated constantly. Showing/hiding certain Views so that it looks like we would have several Pages
- SPA basic understanding needed.
- Single page which is changed based on user actions, by JS code, AJAX requests/responses and react-router routing "going to a new View" with possible routing parameters.
- React components can be:
  - 'Views' = SPA 'pages' we can get routed to
  - Some others are re-usable child components of the Views
- Some are:
  - *container components* who have/fetch the data, hold the state
  - Some others are *presentational components* who get the data from mother, and who only show what they get (plus possibly provide links/buttons related to _that_ item that it's showing)

React routing (SPA routing in frontend, in DOM)

- SPA front-end routing between the Views (~like “going” to a different ‘page’)
- While really just showing and hiding React Views on one downloaded page
- We can also send routing parameter data while going to another View, e.g. id needed by next View
- *How is the react-router routing working in general?*
  - https://youtu.be/Law7wfdg_ls?t=73
  - from 1:13 (link above already at that time) until 16:46 at least.
  - after 16:46 starts "custom routing" i.e. routing with parameters. 33 mins in total.
- **Ingredients of routing**:
  - **react-router-dom** node module,
  - **Router** at root component with **Switch or similar**,
  - **Routes** with **"url" patterns**, possible **parameters**,
  - Each Route mapped to a (‘View’) React **component**,
  - **Link** components in other (view) components' code that use those Routes. Offering navigation to another view

Theming, styling

- *How is the style Theme shared with / provided to all React components?*
  - Answer: Injected to the root React element, and theme-abled child components used, and they automagically read the style from the component tree
- *How is Theme idea good? Fulfills our goal of defining things only once, in one place / file / row etc.*

- Thus the Theme is shared to child componets a bit like the Context is also shared in the component tree!

Frontend architecture principles

- SRP, Single-Responsibility Principle. Each module, function, and item is doing only one thing. (If one module is doing several, split into multiple orchestrated modules)
  - E.g React components should be in their own files so that they can be a) easily reused b) checked fast and with good test need understanding c) seen without scrolling the code
- Cousin of the previous one: Each thing is only specified once, in one place
  - All settings and other constants should be read from env variables or other centralized storage
- Each thing of different nature should be in separate location. Like routing in one folder, all ajax actions in another.
- Each feature (done by a sub-team, feature team) is folder, file, and codeline -wise as independent of others as possible.
  - (codeline-wise: It's not always/mostly possible to add new feature without touching something of the existing shared files. But that should be done as a) as a fast commit and shared to others b) as uniform way as possible c) hopefully in a new codeline separate from the previous items. Seen nicely in Git
    Think about e.g. a new View and its SPA "URL" added to the routing.
- Code should be easy to read by people who did not write it. Maximize reading speed, not writing speed.
- Documentation should be short but informative and easy to modify and commit to git (Markdown or other text format).
  Documentation 'hierarchy': 1. code naming should tell as much it can 2. only then comments in code, 3. only the rest to the separate documentation (in repo Markdown)
  - temp comments in code are good while developing. Use a TODO marker
  - People who do not know the project should be able to install and setup the project. Hand-over to customer is important

Tech used in this semester’s case

See the possible Frontend_UsedTech_20XXY.pdf file in this frontend learning repository (docs_frontend_design).

Other topics for the exam ? (Mainly extras though)

- **AJAX**
- *Something asked about AJAX? If then some core understanding.*
  - *Do we get the result out as JSON text, or already auto-parsed to be a JS object? Depends on library / function.*
  - *Usually each AJAX library has multiple different ways to write exactly same things! (e.g. Axios has, jQuery has,…)*
- **LocalStorage**
  - Browsers are able to save 'text files' to the computer's disk and open them with a key/name e.g. days later. If objects are stringified=(serialized as JSON text) we can even persist (data) objects.
- **Full-stack open 202X, the reading list**
  - has now the green parts that are interesting from Frontend learning point of view!
  - https://github.com/valju/docs_backend_design/blob/master/FSO/FSOReadingList.pdf (in Backend docs repo)
    Note: some things from part 7 might be added for the Spring 2022 exam and onwards for advanced students!
- **For thoughts: Possible frontend project design & creation steps**
  - https://github.com/valju/docs_frontend_design/blob/master/FrontendRelatedSteps_SimilarToOurCases.pdf